

Parallel degree computation for solution space of binomial systems with an application to the master space of $\mathcal{N} = 1$ gauge theories

Tianran Chen

Department of Mathematics, Michigan State University, East Lansing, MI, USA

Dhagash Mehta

Department of Applied and Computational Mathematics and Statistics, University of Notre Dame, Notre Dame, IN 46545, USA

Centre for the Subatomic Structure of Matter, Department of Physics, School of Physical Sciences, University of Adelaide, Adelaide, South Australia 5005, Australia

Abstract

The problem of solving a system of polynomial equations is one of the most fundamental problems in applied mathematics. Among them, the problem of solving a system of binomial equations form a important subclass for which specialized techniques exist. For both theoretic and applied purposes, the degree of the solution set of a system of binomial equations often plays an important role in understanding the geometric structure of the solution set. Its computation, however, is computationally intensive. This paper proposes a specialized parallel algorithm for computing the degree on GPUs that takes advantage of the massively parallel nature of GPU devices. The preliminary implementation shows remarkable efficiency and scalability when compared to the closest CPU-based counterpart. Applied to the “master space problem of $\mathcal{N} = 1$ gauge theories” the GPU-based implementation achieves nearly 30 fold speedup over its CPU-only counterpart enabling the discovery of previously unknown results. Equally important to note is the far superior scalability: with merely 3 GPU devices on a single workstation, the GPU-based implementation shows better performance, on certain problems, than a small cluster totaling 100 CPU cores.

Key words: Binomial Systems, Homotopy Continuation, Algebraic Geometry, BKK Root-count, GPU Computing, Supersymmetric gauge theories.

Email addresses: `chentia1@msu.edu` (Tianran Chen), `dmehta@nd.edu` (Dhagash Mehta).

1. Introduction

The problem of solving a system of polynomial equations is one of the most fundamental problems in applied mathematics and science. Among them, the problem of solving a system of binomial equations is of special interest for they appear naturally in many applications, and specialized and much more efficient algorithm exists (e.g. [25]). In many applications, only the solutions of a system of binomial equations for which no variable is zero are needed. Such solutions are known as the \mathbb{C}^* -solution set and will be the focus of this article.

In this article, we propose a parallel algorithm for computing the degree of a \mathbb{C}^* -solution set of a system of binomial equations. This algorithm is specially designed for GPU (graphics processing unit) devices by taking advantage of the massively parallel nature of GPUs. When applied to a binomial system coming from particle physics, called the master space of $\mathcal{N} = 1$ gauge theories, this algorithm is able to produce previously unknown results. Furthermore, the experimental implementation for GPU built on top of the CUDA framework has already shown promising results. Remarkably, with multiple GPU devices (on the same computer), the GPU based implementation exhibits much better performance, in many cases, than small to medium sized computer clusters.

This article is structured as follows: First, necessary notations and concepts are introduced. In particular, we shall review basic geometric properties of the \mathbb{C}^* -solution set defined by a binomial system. Then the algorithm for computing the Smith Normal Form of an integer matrix is reviewed in §3, as it is an important tool necessary in understanding the structure of the \mathbb{C}^* -solution set of a binomial system. The core of this article is §4 where a highly scalable parallel algorithm for computing the degree of the \mathbb{C}^* -solution set of a system of binomial equations is presented. A natural by-product of the degree computation is a series of homotopy constructions that can be used to compute the “witness sets” of components of the \mathbb{C}^* -solution set of a binomial system, which is an important and ubiquitous construction in numerical algebraic geometry. This process is explained in §5. The problem of studying the master space of $\mathcal{N} = 1$ gauge theories, arising from string theory, is briefly reviewed in §6, and we show the interesting and previously unknown results obtained by applying the parallel algorithm for solving systems of binomial equations and computing the degree of the solution set to the master space problem.

2. Laurent binomial systems and its solution set

First, we shall introduce necessary concepts and notations. For positive integers m and n , let $M_{n \times m}(\mathbb{Z})$ denote the set of all $n \times m$ integer matrices. A square integer matrix is said to be **unimodular** if its determinant is ± 1 . Note that such a matrix $A \in M_{n \times n}(\mathbb{Z})$ has a unique inverse $A^{-1} = \frac{1}{\det A} \text{adj } A$ that is also in $M_{n \times n}(\mathbb{Z})$, where $\text{adj } A$ is the adjoint matrix of A . The $n \times n$ identity matrix in $M_{n \times n}(\mathbb{Z})$ is denoted by I_n .

Even though the main application considered in this article are binomial systems, its theory is more naturally developed in the context of more general “Laurent binomial systems” where negative exponents are allowed. For variables $\mathbf{x} = (x_1, \dots, x_n)$, a **Laurent monomial** in \mathbf{x} is an expression of the form $x_1^{\alpha_1} \cdots x_n^{\alpha_n}$ where $\alpha_1, \dots, \alpha_n$ are integers

(which may be zero or negative). For convenience, we shall write $\alpha = (\alpha_1, \dots, \alpha_n)^\top \in \mathbb{Z}^n$ and use the “vector exponent” notation

$$\mathbf{x}^\alpha = (x_1, \dots, x_n) \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{pmatrix} = x_1^{\alpha_1} \dots x_n^{\alpha_n}$$

to denote a Laurent monomial. Similarly, for an integer matrix $A \in M_{n \times m}(\mathbb{Z})$ with columns $\alpha^{(1)}, \dots, \alpha^{(m)} \in \mathbb{Z}^n$, the “matrix exponent” notation will be used for an m -tuple of Laurent monomials:

$$\mathbf{x}^A = \mathbf{x}^{\left(\alpha^{(1)} \dots \alpha^{(m)}\right)} := (\mathbf{x}^{\alpha^{(1)}}, \dots, \mathbf{x}^{\alpha^{(m)}}). \quad (1)$$

This notation is particularly convenient since the familiar identities $\mathbf{x}^{I_n} = \mathbf{x}$ and $(\mathbf{x}^A)^B = \mathbf{x}^{AB}$ still hold. Since the exponents here may be negative, it is only meaningful to consider the function $\mathbf{x} \mapsto \mathbf{x}^A$ when we restrict each x_i to be nonzero. In particular, throughout this article, we shall let $x_i \in \mathbb{C}^* = \mathbb{C} \setminus \{0\}$ for each $i = 1, \dots, n$. In this case, each matrix $A \in M_{n \times m}(\mathbb{Z})$ induces a function from $(\mathbb{C}^*)^n$ to $(\mathbb{C}^*)^m$ given by $\mathbf{x} \mapsto \mathbf{x}^A$. Of particular importance is the function induced by a unimodular matrix $A \in M_{n \times n}(\mathbb{Z})$ since A^{-1} is also in $M_{n \times n}(\mathbb{Z})$, and hence functions $\mathbf{x} \mapsto \mathbf{x}^A$ and $\mathbf{x} \mapsto \mathbf{x}^{A^{-1}}$ are the inverses of each other $((\mathbf{x}^A)^{A^{-1}} = \mathbf{x}^{AA^{-1}} = \mathbf{x}^{I_n} = \mathbf{x})$.

A **Laurent binomial** is an expression of the form $c_1 \mathbf{x}^\alpha + c_2 \mathbf{x}^\beta$ for some $c_1, c_2 \in \mathbb{C}^*$ and $\alpha, \beta \in \mathbb{Z}^n$. This article focuses on the properties of the solution set of systems of Laurent binomials equations, or simply **Laurent binomial systems**, over $(\mathbb{C}^*)^n$. Stated formally, given exponent vectors $\alpha^{(1)}, \dots, \alpha^{(m)}, \beta^{(1)}, \dots, \beta^{(m)} \in \mathbb{Z}^n$ and the coefficients $c_{i,j} \in \mathbb{C}^*$, the goal is to describe the set of all $\mathbf{x} \in (\mathbb{C}^*)^n$ that satisfies the system of equations

$$\begin{cases} c_{1,1} \mathbf{x}^{\alpha^{(1)}} + c_{1,2} \mathbf{x}^{\beta^{(1)}} & = 0 \\ & \vdots \\ c_{m,1} \mathbf{x}^{\alpha^{(m)}} + c_{m,2} \mathbf{x}^{\beta^{(m)}} & = 0 \end{cases}.$$

Since only the solutions in $(\mathbb{C}^*)^n$ are concerned, this system is clearly equivalent to

$$(\mathbf{x}^{\alpha^{(1)} - \beta^{(1)}}, \dots, \mathbf{x}^{\alpha^{(m)} - \beta^{(m)}}) = (-c_{1,2}/c_{1,1}, \dots, -c_{m,2}/c_{m,1}).$$

With the more compact “matrix exponent” notation in (1), this system can simply be written as

$$\mathbf{x}^A = \mathbf{b} \quad \text{or equivalently} \quad \mathbf{x}^A - \mathbf{b} = \mathbf{0} \quad (2)$$

where the integer matrix $A \in M_{n \times m}(\mathbb{Z})$, having columns $\alpha^{(1)} - \beta^{(1)}, \dots, \alpha^{(m)} - \beta^{(m)}$, represents the exponents appeared in the Laurent monomials and the vector $\mathbf{b} = (-c_{1,2}/c_{1,1}, \dots, -c_{m,2}/c_{m,1})^\top \in (\mathbb{C}^*)^m$ collects all the coefficients. The solution set of (2) over $(\mathbb{C}^*)^n$ shall be denoted by

$$\mathcal{V}^*(\mathbf{x}^A - \mathbf{b}) = \{\mathbf{x} \in (\mathbb{C}^*)^n \mid \mathbf{x}^A - \mathbf{b} = \mathbf{0}\}. \quad (3)$$

The goal of this article is to present efficient parallel algorithms for computing the structural properties of the set $\mathcal{V}^*(\mathbf{x}^A - \mathbf{b})$: its dimension, number of components, global parametrizations, and, most importantly, degree. We shall first briefly review some basic facts about the \mathbb{C}^* -solution set of a Laurent binomial system. A more detailed summary

can be found in the article [6] by the first author and Tien-Yien Li. In depth theoretical discussions can be found in standard references such as [8, 9, 12, 37, 44]. Certain computational aspects have been studied in [25, 26].

An important tool in understanding the structure of $\mathcal{V}^*(\mathbf{x}^A - \mathbf{b})$ is the *Smith Normal Form* of the exponent matrix $A \in M_{n \times m}(\mathbb{Z})$: there are unimodular square matrices $P \in M_{n \times n}(\mathbb{Z})$ and $Q \in M_{m \times m}(\mathbb{Z})$ such that

$$P A Q = \begin{pmatrix} d_1 & & & & \\ & \ddots & & & \\ & & d_r & & \\ & & & 0 & \\ & & & & \ddots \\ & & & & & 0 \end{pmatrix} \in M_{n \times m}(\mathbb{Z}) \quad (4)$$

with nonzero integers $d_1 \mid d_2 \mid \cdots \mid d_r$ for $r = \text{rank } A$, unique up to the signs. Here, $a \mid b$ means a divides b as usual. This decomposition of the matrix A provides important topological information about $\mathcal{V}^*(\mathbf{x}^A - \mathbf{b}) \subset (\mathbb{C}^*)^n$ summarized in the following proposition:

Proposition 1 (Topological description [9]). If $\mathcal{V}^*(\mathbf{x}^A - \mathbf{b})$ in $(\mathbb{C}^*)^n$ is not empty, then it consists of a finite number of connected components. Furthermore,

- (1) the number of components is exactly $\left| \prod_{j=1}^r d_j \right|$.
- (2) each solution component has codimension equal to $\text{rank } A = r$.

This description can be strengthened significantly. Here we shall briefly outline the derivation of the stronger description of $\mathcal{V}^*(\mathbf{x}^A - \mathbf{b})$ as it provides the important data that form the starting point of the degree computation to be discussed in §4. For P and Q in the Smith Normal Form of A in (4), let $P_r \in M_{r \times n}(\mathbb{Z})$ and $P_0 \in M_{(n-r) \times n}(\mathbb{Z})$ be the top r rows and the remaining $n - r$ rows of P respectively. Similarly, let $Q_r \in M_{m \times r}(\mathbb{Z})$ and $Q_0 \in M_{m \times (m-r)}(\mathbb{Z})$ be the left r columns and the remaining $m - r$ columns of Q respectively. With these notations, the Smith Normal Form of (4) of A can be written as

$$\begin{pmatrix} P_r \\ P_0 \end{pmatrix} A \begin{pmatrix} Q_r & Q_0 \end{pmatrix} = \begin{pmatrix} D & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix} \quad (5)$$

with $D = \text{diag}(d_1, \dots, d_r) \in M_{r \times r}(\mathbb{Z})$ and $\mathbf{0}$'s representing zero block matrices of appropriate sizes. With this we can transform the binomial system $\mathbf{x}^A = \mathbf{b}$ into a form from which more detailed information can be easily extracted.

Since P and Q are both unimodular the maps $\mathbf{z} \mapsto \mathbf{z}^P$ and $\mathbf{y} \mapsto \mathbf{y}^Q$ are both bijections on $(\mathbb{C}^*)^n$ and $(\mathbb{C}^*)^m$ respectively. Therefore, considering the solution set in $(\mathbb{C}^*)^n$, the original system $\mathbf{x}^A = \mathbf{b}$ is equivalent to $(\mathbf{x}^A)^Q = \mathbf{x}^{A Q} = \mathbf{b}^Q$. Similarly, the solution sets remain equivalent after the change of variables $\mathbf{x} = \mathbf{z}^P$, which produces

$$(\mathbf{z}^P)^{A Q} = \mathbf{z}^{P A Q} = \mathbf{z}^{\begin{pmatrix} D & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix}} = (\mathbf{z}^{\begin{pmatrix} D \\ \mathbf{0} \end{pmatrix}}, \mathbf{z}^{\begin{pmatrix} \mathbf{0} \\ \mathbf{0} \end{pmatrix}}) = \mathbf{b}^Q = (\mathbf{b}^{Q_r}, \mathbf{b}^{Q_0}).$$

Since $D = \text{diag}(d_1, \dots, d_r) \in M_{r \times r}(\mathbb{Z})$, the original system $\mathbf{x}^A = \mathbf{b}$ can now be decomposed into a combined system

$$(z_1, \dots, z_r) \begin{pmatrix} d_1 & & \\ & \ddots & \\ & & d_r \end{pmatrix} = \mathbf{b}^{Q_r} \quad (6)$$

$$\mathbf{1} = \mathbf{b}^{Q_0} \quad (7)$$

$$z_{r+1}, \dots, z_n : \text{ free} \quad (8)$$

where (7) appears when $r < m$ with $\mathbf{1} = (1, \dots, 1) \in (\mathbb{C}^*)^{m-r}$, and (8) appears when $r < n$. The word “free” in (8) means the system imposes no constraints on the $n - r$ variables z_{r+1}, \dots, z_n .

Focusing on the above decomposed system, it is clear that if $r < m$, then the system is inconsistent unless $\mathbf{1} = \mathbf{b}^{Q_0}$. If the system is consistent (namely, (7) holds), then the solutions to (6) are exactly

$$\begin{cases} z_1 = e^{2k_1\pi/d_1}\zeta_1 & \text{for } k_1 = 0, \dots, d_1 - 1 \\ z_2 = e^{2k_2\pi/d_2}\zeta_2 & \text{for } k_2 = 0, \dots, d_2 - 1 \\ \vdots & \vdots \\ z_r = e^{2k_r\pi/d_r}\zeta_r & \text{for } k_r = 0, \dots, d_r - 1 \end{cases} \quad (9)$$

where each ζ_j is a fixed choice of the d_j -th root of j -th coordinate of \mathbf{b}^Q . Clearly, all of them are isolated and the total number of these solutions is $|\prod_{j=1}^r d_j| = |\det D|$. If $r < n$, then the solution set of the decomposed system (6)–(8) in $(\mathbb{C}^*)^n$ breaks into “components” of the form $\{(e^{2k_1\pi/d_1}\zeta_1, \dots, e^{2k_r\pi/d_r}\zeta_r, z_{r+1}, \dots, z_n) : (z_{r+1}, \dots, z_n) \in (\mathbb{C}^*)^{n-r}\}$, and they are in one-to-one correspondence with solutions in (9). Since each component is parametrized by the $n - r$ free variables z_{r+1}, \dots, z_n , it is smooth and of dimension $n - r$. Furthermore, they are disjoint, because these components have distinct z_1, \dots, z_r coordinates.

To translate the above description of the $(\mathbb{C}^*)^n$ -solution set of the decomposed system (in \mathbf{z}) into a description the original solution set $\mathcal{V}^*(\mathbf{x}^A - \mathbf{b})$, one may simply apply the change of variables $\mathbf{x} = \mathbf{z}^P$. Note that this map and its inverse $\mathbf{z} = \mathbf{x}^{P^{-1}}$ are both given by monomials (*bi-regular* maps [21]), the basic properties of the solution set, such as, the number of solution components, their dimensions, and smoothness are therefore preserved. To summarize, the above elaborations assert the following proposition.

Proposition 2 (Global parametrization [9, 25, 44]). For the solution set $\mathcal{V}^*(\mathbf{x}^A - \mathbf{b})$ in $(\mathbb{C}^*)^n$, let P, Q, Q_0 and D be those matrices appeared in the decompositions of A in (4) and (5), and let $r = \text{rank } A$.

If $\mathbf{1} \neq \mathbf{b}^{Q_0}$ then the binomial system is inconsistent, and hence its solution set in $(\mathbb{C}^*)^n$ is empty.

If $\mathbf{1} = \mathbf{b}^{Q_0}$ then the solution set of $\mathbf{x}^A = \mathbf{b}$ in $(\mathbb{C}^*)^n$ consists of $|\prod_{j=1}^r d_j| = |\det D|$ connected components V_{k_1, \dots, k_r} for $k_1 \in \{0, \dots, d_1 - 1\}, \dots, k_r \in \{0, \dots, d_r - 1\}$. Each component V_{k_1, \dots, k_r} is smooth of dimension $n - r$, and it is parametrized by the smooth global parametrization $\phi_{k_1, \dots, k_r} : (\mathbb{C}^*)^{(n-r)} \rightarrow V_{k_1, \dots, k_r}$ given by

$$\phi_{k_1, \dots, k_r}(t_1, \dots, t_{n-r}) = (e^{2k_1\pi/d_1}\zeta_1, \dots, e^{2k_r\pi/d_r}\zeta_r, t_1, \dots, t_{n-r})^P \quad (10)$$

where each ζ_j is a fixed choice of the d_j -th root of the j -th coordinate of \mathbf{b}^Q .

Note that, as previously stated, in the case of $r = n$, the solution set $\mathcal{V}^*(\mathbf{x}^A - \mathbf{b})$ is of dimension $n - r = 0$, that is, $\mathcal{V}^*(\mathbf{x}^A - \mathbf{b})$ consists of isolated points. Then the “parametrizations” ϕ_{k_1, \dots, k_r} are understood as constants each describes a single isolated point.

As indicated in Proposition 2, for a consistent Laurent binomial system $\mathbf{x}^A = \mathbf{b}$ where $A \in M_{n \times m}(\mathbb{Z})$ with $r = \text{rank}(A) < n$, each component of the solution set in $(\mathbb{C}^*)^n$ will be of dimension $n - r > 0$. In this situation, for both theoretical interests and demands from concrete applications, like the Master Space problem to be discussed in §6, one often wishes to identify another important property: the *degrees* of the components. Degree is a classic concept developed for plane algebraic curves. For example, the quadratic equation $y - x^2 = 0$ defines a curve of degree 2, i.e., the parabola. The generalized notation of degree for irreducible algebraic sets is usually formulated algebraically via Hilbert Polynomials. In this article, following the common practice of Numerical Algebraic Geometry, we shall take a geometric approach: Let $V = V_{k_1, \dots, k_r}$ be a component of $\mathcal{V}^*(\mathbf{x}^A - \mathbf{b})$ for some fixed choice of k_1, \dots, k_n as defined in Proposition 2. The number of isolated intersection point between V and a “generic” affine space of complementary dimension is a fixed number, and this number is the **degree** of V , denoted by $\deg V$. In algebraic terms, we are considering the degree of the *projective closure* of V .

Stated more precisely, let \mathcal{G}_r be the set of all affine space in \mathbb{C}^n of dimension $r = n - \dim V$. Then it can be shown that in a fixed open and dense subset of \mathcal{G}_r , all the affine spaces intersect with V at a fixed number of isolated points. This geometric interpretation of degree is explained in [13, 21, 43].

From a computational standpoint, a generic affine space in \mathcal{G}_r can be represented by the solution set of a system of $d := n - r$ linear equations with generic coefficients. Therefore $\deg V$ is precisely the number of points $\mathbf{x} = (x_1, \dots, x_n) \in V$ that satisfies the system of linear equations

$$\begin{cases} c_{11}x_1 + c_{12}x_2 + \dots + c_{1n}x_n &= c_{10} \\ &\vdots \\ c_{d1}x_1 + c_{d2}x_2 + \dots + c_{dn}x_n &= c_{d0}. \end{cases} \quad (11)$$

where c_{ij} for $i = 1, \dots, d$ and $j = 1, \dots, n$ are generic complex numbers. But recall that the set $V = V_{k_1, \dots, k_r}$ is precisely the image of the injective map

$$\phi_{k_1, \dots, k_r}(t_1, \dots, t_d) = (e^{2k_1\pi/d_1}\zeta_1, \dots, e^{2k_r\pi/d_r}\zeta_r, t_1, \dots, t_d)^P$$

in Proposition 2. If we let $\xi = (e^{2k_1\pi/d_1}\zeta_1, \dots, e^{2k_r\pi/d_r}\zeta_r)$ and $\mathbf{t} = (t_1, \dots, t_d)$ then

$$\phi_{k_1, \dots, k_r}(\mathbf{t}) = (\xi, \mathbf{t}) \begin{pmatrix} P_r \\ P_0 \end{pmatrix} = (\xi^{\mathbf{p}_r^{(1)}} \mathbf{t}^{\mathbf{p}_0^{(1)}}, \dots, \xi^{\mathbf{p}_r^{(n)}} \mathbf{t}^{\mathbf{p}_0^{(n)}})$$

where for each $j = 1, \dots, n$, $\mathbf{p}_r^{(j)}$ and $\mathbf{p}_0^{(j)}$ are the j -th columns of P_r and P_0 respectively. In other words, V has the global parametrization $x_i = \xi^{\mathbf{p}_r^{(1)}} \mathbf{t}^{\mathbf{p}_0^{(1)}}$. Therefore the intersections between V and the generic affine space defined by (11) are precisely the solutions of the polynomial system

$$\begin{cases} c_{11} \xi^{\mathbf{p}_r^{(1)}} \mathbf{t}^{\mathbf{p}_0^{(1)}} + c_{12} \xi^{\mathbf{p}_r^{(2)}} \mathbf{t}^{\mathbf{p}_0^{(2)}} + \dots + c_{1n} \xi^{\mathbf{p}_r^{(n)}} \mathbf{t}^{\mathbf{p}_0^{(n)}} &= c_{10} \\ &\vdots \\ c_{d1} \xi^{\mathbf{p}_r^{(1)}} \mathbf{t}^{\mathbf{p}_0^{(1)}} + c_{d2} \xi^{\mathbf{p}_r^{(2)}} \mathbf{t}^{\mathbf{p}_0^{(2)}} + \dots + c_{dn} \xi^{\mathbf{p}_r^{(n)}} \mathbf{t}^{\mathbf{p}_0^{(n)}} &= c_{d0} \end{cases}$$

By letting $c'_{ij} := c_{ij}\xi^{\mathbf{p}_0^{(j)}} \in \mathbb{C}$ and $c'_{i0} = c_{i0}$ for each $i = 1, \dots, d$ and $j = 1, \dots, n$, the above is a systems of d polynomial equations in variables $\mathbf{t} = (t_1, \dots, t_d)$ with generic complex coefficients c'_{ij} and the same set of monomials $\mathbf{t}^{\mathbf{p}_0^{(j)}}$.

Proposition 3 (Degree via affine space cut). If $r < n$ and $\mathcal{V}^*(\mathbf{x}^A - \mathbf{b}) \neq \emptyset$, then the degree of each component V of $\mathcal{V}^*(\mathbf{x}^A - \mathbf{b})$ agrees with the number of solutions $\mathbf{t} \in (\mathbb{C}^*)^d$ of the system of d Laurent polynomial equations

$$\begin{cases} c_{11}\mathbf{t}^{\mathbf{p}_0^{(1)}} + c_{12}\mathbf{t}^{\mathbf{p}_0^{(2)}} + \dots + c_{1n}\mathbf{t}^{\mathbf{p}_0^{(n)}} &= c_{10} \\ &\vdots \\ c_{d1}\mathbf{t}^{\mathbf{p}_0^{(1)}} + c_{d2}\mathbf{t}^{\mathbf{p}_0^{(2)}} + \dots + c_{dn}\mathbf{t}^{\mathbf{p}_0^{(n)}} &= c_{d0} \end{cases} \quad (12)$$

for generic complex coefficients $c_{ij} \in \mathbb{C}$.

It is important to note that for generic coefficients, the \mathbb{C}^* -solutions of the above system are all isolated (0-dimensional), and the total number is a constant. Indeed, in [27], Kushnirenko has shown that this number can be expressed in terms of the volume of a geometric object: the *Newton polytope* of the above system. Here we state the result in the context of degree computation and leave the technical statement of Kushnirenko's theorem, as well as its related concepts to Appendix §A.

Proposition 4 (Degree as volume). For generic choices of the coefficients, the number of solutions $\mathbf{t} \in (\mathbb{C}^*)^d$ of the system of Laurent polynomial equation (12) and hence the degree of V is

$$\deg V = d! \cdot \text{Vol}_d(\text{conv}\{\mathbf{p}_0^{(1)}, \dots, \mathbf{p}_0^{(n)}, \mathbf{0}\}) \quad (13)$$

where $\mathbf{0} = (0, \dots, 0)^\top \in \mathbb{R}^d$ and columns $\mathbf{p}_0^{(1)}, \dots, \mathbf{p}_0^{(n)}$ of the matrix P_0 are considered as points in \mathbb{R}^d . The notation conv denotes the operation of taking convex hull, and Vol_d is the volume of a convex body in \mathbb{R}^d .

The degree of the solution set of $\mathbf{x}^A = \mathbf{b}$ can be computed efficiently through methods in combinatorial geometry.

3. Parallel Smith Normal Form computation

As summarized in Equation 2, the key to finding the dimension, number of components, and global parametrization of the \mathbb{C}^* -solution set $\mathcal{V}^*(\mathbf{x}^A - \mathbf{b}) \subset (\mathbb{C}^*)^n$ is the Smith Normal Form (4) of the exponent matrix A . In this section, we briefly review the procedure for computing the Smith Normal Form of an integer matrix and then outline the parallel modification that is suitable for both multi-core systems and GPU.

We first briefly review the standard algorithm for computing the Smith Normal Form of a matrix with integer entries. One of the classic algorithms for computing the Smith Normal Form uses successive row (n) and column (m) reductions of the input matrix, as listed in [7, Algorithm 2.4.14] and [16, Section 8.5.1]: Consider the special case where $A = \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}$ with a_1, a_2 nonzero, that is, take $n = 2$ and $m = 1$. By the Bézout's identity, there exist s and t such that $d := \gcd(a_1, a_2) = s a_1 + t a_2$. Let

$$P = \begin{pmatrix} s & t \\ -\frac{a_2}{d} & \frac{a_1}{d} \end{pmatrix},$$

Row length	Speedup ratio
10	0.00
50	0.00
100	1.91
200	1.99
400	8.01
800	14.20
1600	22.00
3200	31.79

Table 1. Speedup ratio achieved by a NVidia GTX 780 graphics card (GPU) on a *single* row reduction operation on when compared to an equivalent single-threaded CPU-based implementation on a Intel Core i7 4770k CPU. In each case, the speedup ratio is computed as the average of three runs. Time consumed by data transfer is *not* computed.

then $\det P = \frac{sa_1 + ta_2}{d} = \frac{d}{d} = 1$ and

$$PA = \begin{pmatrix} s & t \\ -\frac{a_2}{d} & \frac{a_1}{d} \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} sa_1 + ta_2 \\ -\frac{a_2a_1}{d} + \frac{a_2a_2}{d} \end{pmatrix} = \begin{pmatrix} d \\ 0 \end{pmatrix}$$

Similarly, for the special case $A = \begin{pmatrix} a_1 & a_2 \end{pmatrix}$, let $Q = \begin{pmatrix} s & -\frac{a_2}{d} \\ t & \frac{a_1}{d} \end{pmatrix}$, then $AQ = \begin{pmatrix} d & 0 \end{pmatrix}$. In general, $n \times n$ and $m \times m$ version of the above matrices P and Q can be constructed to perform row and column reduction respectively for a $n \times m$ integer matrix.

After repeated such row and column reduction together with potential row and column permutations one can construct unimodular matrices $P^{(1)}, \dots, P^{(k)} \in M_{n \times n}(\mathbb{Z})$ and $Q^{(1)}, \dots, Q^{(\ell)} \in M_{m \times m}(\mathbb{Z})$ such that

$$P^{(k)} \dots P^{(1)} A Q^{(1)} \dots Q^{(\ell)} = \begin{pmatrix} d_1 & & & & \\ & \ddots & & & \\ & & d_r & & \\ & & & 0 & \\ & & & & \ddots \\ & & & & & 0 \end{pmatrix}$$

with $r = \text{rank } A$ and d_1, \dots, d_r nonzero. As noted in standard references such as [16], further reduction can ensure $d_1 \mid d_2 \mid \dots \mid d_r$, but for the purpose of solving binomial system it is not necessary.

By their design, GPUs are naturally well suited to perform the row and column reductions [40] used in computing the Smith Normal Form. As Table 1 shows, GPUs have a clear advantage over CPUs in performing simple row reductions for sufficiently large matrices: over 30 fold speedup can be achieved. §6 shows the result of this algorithm when applied to the master space problem.

4. Parallel degree computation

When the solution set consists of positive dimensional components, Proposition 4 provides a computationally viable means for computing the degree of each component as the normalized volume of a convex polytope. In this section we shall present a parallel algorithm for computing the degree that is suitable for both multi-core systems and GPUs, though the focus is the GPU-based implementation. Throughout this section, let V be a component of $\mathcal{V}^*(\mathbf{x}^A - \mathbf{b}) \subset (\mathbb{C}^*)^n$, and let $d = \dim V = n - r$. Here we shall focus on the case where $d > 0$. Let $P^0 = (\mathbf{p}_0^{(1)}, \dots, \mathbf{p}_0^{(n)}) \in M_{d \times n}(\mathbb{Z})$ be the matrix appears in the Smith Normal Form of A (4). Considering each $\mathbf{p}_0^{(j)}$ as a point in \mathbb{R}^d , let $S = \{\mathbf{p}_0^{(1)}, \dots, \mathbf{p}_0^{(n)}\} \subset \mathbb{R}^d$ be the finite point set. Then by Proposition 4,

$$\deg V = d! \text{Vol}_d(\text{conv } S). \quad (14)$$

For brevity, let $\text{NVol}_d = d! \text{Vol}$ be the **normalized volume** function in \mathbb{R}^d , then the above equation can be written as

$$\deg V = \text{NVol}_d(\text{conv } S) \quad (15)$$

Therefore the task of computing the degree for V is equivalent to the computation of the normalized volume of a *lattice polytope* (a polytope whose vertices have integer coordinates).

Remark 1. Clearly, (14) and (15) are equivalent. However, from the computational point of view, there is one crucial distinction: The knowledge that $\text{NVol}_d(S)$ must be an integer permits the use of efficient but potentially less accurate numerical methods using floating point arithmetic and still obtain the correct result. Indeed, the exact results can still be obtained as long as the total absolute error is kept below $1/2$. This is not possible for methods that are designed to compute volume of more general polytopes. The algorithm, for computing (15), to be presented below, is hence not directly comparable to *exact volume computation* algorithms [2, 3] for general polytopes.

Our parallel algorithm for the degree computation is developed based on the parallel “mixed cell enumeration” algorithm presented in [5]. (See Remark 2 below) Among many different approaches for computing the normalized volume, here we adopt a technique known as *regular simplicial subdivision* [31]. This approach produces an important byproduct that will be used in the computation of witness set, which will be the subject of §5. In this approach, we are interested in computing the normalized volume $\text{NVol}_d(\text{conv } S)$ by dividing the lattice polytope $\text{conv } S$ into a collection of smaller pieces for which the volume computation is easy.

Definition 1. A **cell** of S is simply an affinely independent subset of S . A **simplicial subdivision** of S is a collection \mathcal{D} of cells of S , such that

- (1) For each $C \in \mathcal{D}$, $\text{conv } C$ is a d -simplex inside $\text{conv } S$;
- (2) For any distinct pair of simplices $C_1, C_2 \in \mathcal{D}$, the intersection of $\text{conv } C_1$ and $\text{conv } C_2$, if nonempty, is a common face of the two; and
- (3) The union of convex hulls of all cells in \mathcal{D} is exactly $\text{conv } S$.

A simplicial subdivision plays an important role in computing $\text{NVol}_d(\text{conv } S)$: the normalized volume of a d -simplex in \mathbb{R}^d is easy to compute: given a d -simplex $\Delta = \text{conv}\{\mathbf{a}_0, \dots, \mathbf{a}_d\} \subset \mathbb{Z}^d$,

$$\text{NVol}_d(\Delta) = \det \begin{pmatrix} \mathbf{a}_0 & \dots & \mathbf{a}_d \end{pmatrix}. \quad (16)$$

So the volume of $\text{conv } S$ can be computed easily as the sum of the volume of all simplices in \mathcal{D} .

Note that the simplicial subdivision for a given polytope is, in general, not unique, and there are many different approaches for constructing them. Here we focus on the approach of regular simplicial subdivision: One can define a “lifting function” $\omega : S \rightarrow \mathbb{R}$ by assigning a real number to each point in S . For each point $\mathbf{a} \in S$, a new point $(\mathbf{a}, \omega(\mathbf{a})) \in \mathbb{R}^{d+1}$ can be created by using $\omega(\mathbf{a})$ as an additional coordinate. This procedure “lifts” points of S into \mathbb{R}^{d+1} , the space of one higher dimension. Let

$$\hat{S} = \{\hat{\mathbf{a}} = (\mathbf{a}, \omega(\mathbf{a})) \mid \mathbf{a} \in S\} \quad (17)$$

be the lifted version of S via the lifting function ω . Figures 1a and 1b show examples of this lifting procedure.. Let $\pi : \mathbb{R}^{d+1} \rightarrow \mathbb{R}^d$ be the projection that simply erases the last coordinate, then $\pi(\hat{S}) = S$.

Recall that for a face \hat{F} of the lifted polytope $\text{conv } \hat{S}$, its *inner normal* is a vector $\hat{\boldsymbol{\alpha}} \in \mathbb{R}^{d+1}$ such that the linear functional $\langle \bullet, \hat{\boldsymbol{\alpha}} \rangle$ attains its minimum over $\text{conv } \hat{S}$ on \hat{F} . Moreover, a face \hat{F} of $\text{conv } \hat{S}$ is called a **lower face** with respect to the projection π if its inner normal $\hat{\boldsymbol{\alpha}}$ has positive last coordinate. Without loss of generality, in this case, we may assume the last coordinate of $\hat{\boldsymbol{\alpha}}$ to be 1, that is, $\hat{\boldsymbol{\alpha}} = (\alpha_1, \dots, \alpha_n, 1) \in \mathbb{R}^{d+1}$. It can be shown that for *almost all* choices of the lifting function $\omega : S \rightarrow \mathbb{R}$, the projections of all the d -dimensional lower faces of $\text{conv } \hat{S}$ via π form a simplicial subdivision for $\text{conv } S$ which is called a *regular simplicial subdivision* of $\text{conv } S$. The construction of this simplicial subdivision is therefore equivalent to the enumeration of all the lower faces of $\text{conv } \hat{S}$.

Example 1. Consider, for example, $S = \{(0,0), (0,1), (1,1), (1,0)\}$ in the xy -plane. A simplicial subdivision of $\text{conv } S$ can be obtained via the following procedure: First one assign “liftings” $\omega_1, \omega_2, \omega_3, \omega_4 \in \mathbb{R}$ to each of the vertices as the z -coordinate and obtain new points $(0,0,\omega_1), (0,1,\omega_2), (1,1,\omega_3), (1,0,\omega_4)$ in \mathbb{R}^3 . It is easy to verify that with almost all choices of the liftings the four “lifted” points (Figure 1a) do not lie on the same plane. In that case, the convex hull $\text{conv}\{(0,0,\omega_1), (0,1,\omega_2), (1,1,\omega_3), (1,0,\omega_4)\}$ of the four lifted vertices form a three dimensional polytope (Figure 1b) with triangle faces. Of particular importance is the lower hull of this polytope which are the faces facing downward. As shown in Figure 1c, the projection of the faces in the lower hull back onto the xy -plane form a simplicial subdivision of the original shape $\text{conv } S$.

Algebraically speaking, a d -dimensional lower face of $\text{conv } S$ is the convex hull of a set of $d+1$ points $\{\hat{\mathbf{a}}_0, \dots, \hat{\mathbf{a}}_d\} \subset \hat{S}$ for which there exists a $\hat{\boldsymbol{\alpha}} = (\boldsymbol{\alpha}, 1) \in \mathbb{R}^{d+1}$ such that the system of inequalities

$$I(\mathbf{a}_0, \dots, \mathbf{a}_d) : \begin{cases} \langle \hat{\mathbf{a}}_0, \hat{\boldsymbol{\alpha}} \rangle = \langle \hat{\mathbf{a}}_j, \hat{\boldsymbol{\alpha}} \rangle & \text{for } j = 1, \dots, d \\ \langle \hat{\mathbf{a}}_0, \hat{\boldsymbol{\alpha}} \rangle \leq \langle \hat{\mathbf{a}}, \hat{\boldsymbol{\alpha}} \rangle & \text{for } \mathbf{a} \in S \end{cases} \quad (18)$$

is satisfied. In other words, the existence of the lower face defined by $\{\hat{\mathbf{a}}_0, \dots, \hat{\mathbf{a}}_d\} \subset \hat{S}$ is equivalent to the feasibility of the above system of inequalities $I(\mathbf{a}_0, \dots, \mathbf{a}_d)$. This

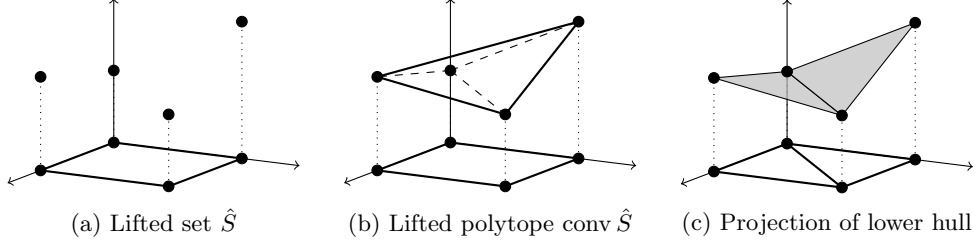


Fig. 1. Regular simplicial subdivision via generic lifting

algebraic description of the lower faces is the basis on which enumeration methods are developed. While a brute-force approach of checking all the possible combinations of $d+1$ points in S against the system of inequalities (18) may be possible, the combinatorial explosion will likely render it impractical for all but the most trivial cases.

In the following subsections, we shall present an approach that results in a *parallel algorithm* which is suitable for both multi-core systems and GPU devices. In this approach, we employ two complementing processes of “extension” and “pivoting”. We shall outline them below.

Remark 2 (Connection to existing works). The approach developed in this article is a natural continuation of a rich web of works on the “mixed cell enumeration” problem initiated by the seminal work [24]. The degree computation problem can be considered as a special case of the mixed cell enumeration problem, and the connection is explained in §A. Active development in the algorithmic aspects of this problem can be found in works such as [5, 14, 15, 28, 30, 38, 39, 45]. A broad survey of this topic can be found in [29].

The “pivoting” process (for “mixed cell enumeration”), to be described below, was proposed in [14]. However, in terms of performance, it was quickly eclipsed by the “extension” process developed in [30], [15], and [38, 39]. In the present article, the pivoting and the extension processes are combined as we believe the complementing duo offers much better scalability which is crucial in the GPU-based implementations. This is confirmed by the numerical experiments, to be presented in §6.

The graph-theoretic view of the “cell enumeration” process, adopted in this article, was originally developed in [14] and, independently, in [39]. The parallelization of the algorithm follows the same general idea attempted in [5], but it is modified, in this article, to adapt to the massively parallel GPU architectures.

4.1. Extension of k -faces

Intuitively speaking, in the extension process, one starts with the vertices of the lower hull of $\text{conv } \hat{S}$. For each of these vertices, systematic attempts are made to “extend” it by finding another lower vertex so that the two vertices form a “lower edge” (an edge on the lower hull of $\text{conv } \hat{S}$). The possible extensions may not be unique, and for each possibility, further attempts are made to extend it to 2-dimensional lower faces. This process continues until one reaches all the d -dimensional lower faces. Finally, the collection of such d -dimensional lower faces will project down, via π , to form a simplicial subdivision for $\text{conv } S$.

To describe this process, we first extend the characterization (18) to include lower faces of all dimensions: A set of affinely independent $k+1$ points in \hat{S} is said to determine

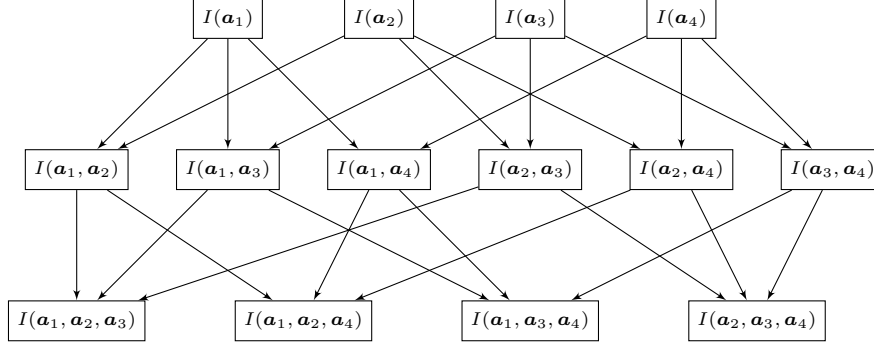


Fig. 2. A directed acyclic graph of possible lower k -faces

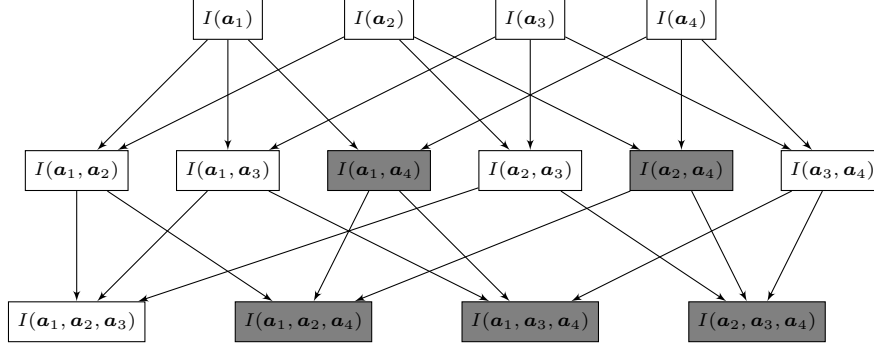


Fig. 3. A direct graph of possible lower k -faces colored by the feasibility of the corresponding system of inequalities.

a **lower k -face** if their convex hull form a k -dimensional lower face of $\text{conv } \hat{S}$ with respect to the projection π . Stated algebraically, the affinely independent set $\{\mathbf{a}_0, \dots, \mathbf{a}_k\}$ determines a lower k -face if and only if there exists an $\hat{\alpha} = (\alpha, 1) \in \mathbb{R}^{d+1}$, such that the system of inequalities

$$I(\mathbf{a}_0, \dots, \mathbf{a}_k) : \begin{cases} \langle \hat{\mathbf{a}}_0, \hat{\alpha} \rangle = \langle \hat{\mathbf{a}}_j, \hat{\alpha} \rangle & \text{for } j = 1, \dots, k \\ \langle \hat{\mathbf{a}}_0, \hat{\alpha} \rangle \leq \langle \hat{\mathbf{a}}, \hat{\alpha} \rangle & \text{for } \mathbf{a} \in S \end{cases} \quad (19)$$

is satisfied.

Clearly, a lower 0-face is a vertex on the lower hull of $\text{conv } \hat{S}$. Similarly, a lower 1-face is simply a lower edge. We can conveniently organize all possible system of inequalities of the above form into a *directed acyclic graph*, as illustrated in Figure 2, where each node represents a system of inequalities and there is an edge from $I(A)$ to $I(B)$ whenever B is obtained by joining a new points in S into A . With this construction, the resulting graph is *graded* by the number of points involved.

It can be easily verified that for generic lifting function ω , containment relation between lower k -faces of the same dimension is impossible. That is, for a fixed k , no lower k -face is contained in another lower k -face. Therefore the the graph describes precisely the containment relationship among possible lower k -faces.

A node is said to be feasible if the corresponding system of inequality is feasible. Figure 3 shows an example of the labeling of the graph via the feasibility of the nodes:

dark for infeasible nodes and white for feasible ones. Recall that a node determines to lower k -faces if and only if it is feasible. Hence we only need to explore of the feasible subgraph (the white subgraph in Figure 3).

One crucial observation is that if two points do not define a lower edge, then they cannot be a part of any lower faces of dimension greater than 1. More generally, if a set of points does not define a lower k -face, then there are no lower j -faces containing them for any $j > k$. Stated formally, for $\hat{F}_1 \subset \hat{S}$,

$$I(\hat{F}_1) \text{ is infeasible} \implies I(\hat{F}) \text{ is infeasible for all } \hat{F}_1 \subset \hat{F} \subset \hat{S}. \quad (20)$$

In terms of the graph, if a node is infeasible, then the entire subgraph reachable by that node is infeasible.

Therefore during the exploration of the graph, once an infeasible node is encountered, no further exploration from that node is needed as all nodes reachable are infeasible. This simple observation produces significant savings in terms of computation.

A key procedure in the exploration of the feasible subgraph is the jump from one feasible node to another along an edge. Assuming, for some $\{\mathbf{a}_0, \dots, \mathbf{a}_k\} \subset S$, the node $I(\mathbf{a}_0, \dots, \mathbf{a}_k)$ is feasible, then the feasibility of an adjacent node, say via the edge \mathbf{a}_{k+1} , can be determined by solving the *linear programming problem*

$$\begin{aligned} & \text{Minimize } \langle \hat{\mathbf{a}}_{k+1}, \hat{\boldsymbol{\alpha}} \rangle - \langle \hat{\mathbf{a}}_0, \hat{\boldsymbol{\alpha}} \rangle \text{ subject to} \\ LP(\mathbf{a}_0, \dots, \mathbf{a}_k; \mathbf{a}_{k+1}) : & \quad \langle \hat{\mathbf{a}}_0, \hat{\boldsymbol{\alpha}} \rangle = \langle \hat{\mathbf{a}}_j, \hat{\boldsymbol{\alpha}} \rangle \quad \text{for } j = 1, \dots, k \\ & \quad \langle \hat{\mathbf{a}}_0, \hat{\boldsymbol{\alpha}} \rangle \leq \langle \hat{\mathbf{a}}, \hat{\boldsymbol{\alpha}} \rangle \quad \text{for all } \mathbf{a} \in S \end{aligned} \quad (21)$$

with the variable $\hat{\boldsymbol{\alpha}} = (\boldsymbol{\alpha}, 1)$ for $\boldsymbol{\alpha} \in \mathbb{R}^d$.

Note that under the constraints, the value of the objective function must be non-negative. Indeed, the minimum value of 0 is attainable precise when there is an $\hat{\boldsymbol{\alpha}}$ for which the constraints are satisfied, simultaneously to $\langle \hat{\mathbf{a}}_{k+1}, \hat{\boldsymbol{\alpha}} \rangle = \langle \hat{\mathbf{a}}_0, \hat{\boldsymbol{\alpha}} \rangle$. That is, minimum value is 0 if and only if $I(\mathbf{a}_0, \dots, \mathbf{a}_k, \mathbf{a}_{k+1})$ is feasible. In this case, a new node $I(\mathbf{a}_0, \dots, \mathbf{a}_k, \mathbf{a}_{k+1})$ is discovered. Geometrically, we have “extended” the lower k -face determined by $\{\hat{\mathbf{a}}_0, \dots, \hat{\mathbf{a}}_k\}$ into a lower $(k+1)$ -face by joining it the new vertex $\hat{\mathbf{a}}_{k+1}$.

Using the extension procedure as a basic building block, we shall discuss, in §4.3, we shall discuss the complete parallel algorithm for the exploration of the feasible subgraph.

4.2. Simplicial pivoting

In the above we have described a process that gradually explore the feasible subgraph via extension procedures. This process is complemented by another process which we shall call “simplicial pivoting” which explore the feasible subgraph by “moving sideways” in the graph from one lower d -face to another.

This process starts with a lower d -face of $\text{conv } \hat{S}$ already obtained. Consider, for example, one of the lower face shown in Figure 4. Using an edge as a hinge, we shall “pivot” one lower face until another lower face is obtained. More generally, recall that a lower d -face is determined by a set of $d+1$ affinely independent points $\{\hat{\mathbf{a}}_0, \dots, \hat{\mathbf{a}}_d\}$ in \hat{S} that has an

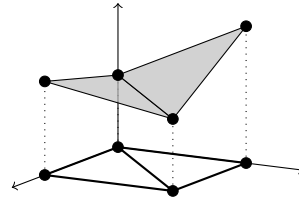


Fig. 4: Simplicial pivoting procedure moves from one lower face to another

inner normal of the form $\hat{\alpha} = (\alpha, 1)$ with $\alpha \in \mathbb{R}^{d+1}$. Stated algebraically, the system of inequalities

$$\begin{aligned} \langle \hat{a}_0, \hat{\alpha} \rangle &= \langle \hat{a}_j, \hat{\alpha} \rangle & \text{for } j = 1, \dots, d \\ \langle \hat{a}_0, \hat{\alpha} \rangle &\leq \langle \hat{a}, \hat{\alpha} \rangle & \text{for all } a \in S \end{aligned} \quad (22)$$

is satisfied. Note that the d equalities form a system of d linearly independent constraints on $\alpha \in \mathbb{R}^d$ and hence uniquely determines α . By removing a single equality from the above system, we give the inner normal $\hat{\alpha}$ one degree of freedom which would allow it to “pivot”. The goal is to let it pivot until it defines a different lower d -face.

For any choice $i = 0, \dots, d$, with the equality corresponding to \hat{a}_i in the above system (22) removed, the inner normal $\hat{\alpha} = (\alpha, 1) \in \mathbb{R}^d$, now with one degree of freedom, is characterized by the system

$$P(a_0, \dots, a_d; i) : \begin{cases} \langle \hat{a}_0, \hat{\alpha} \rangle = \langle \hat{a}_j, \hat{\alpha} \rangle & \text{for } j = 1, \dots, d, \text{ but } j \neq i \\ \langle \hat{a}_0, \hat{\alpha} \rangle \leq \langle \hat{a}_i, \hat{\alpha} \rangle \\ \langle \hat{a}_0, \hat{\alpha} \rangle \leq \langle \hat{a}, \hat{\alpha} \rangle & \text{for all } a \in S \end{cases} \quad (23)$$

Note that this system has $d - 1$ equalities. If a solution with d equalities exists, then that solution corresponds to a different lower d -face. In the context of Linear Programming, such a solution is called a *basic feasible solution*. The problem of finding a basic feasible solution is known as the *Phase One* problem in Linear Programming. It can be solved exactly and efficiently.

This procedure is called “simplicial pivoting”. It allows us to pivot from one lower d -face to another. By repeatedly applying this procedure, more lower d -faces can be gathered. Figure 5 illustrates this process.

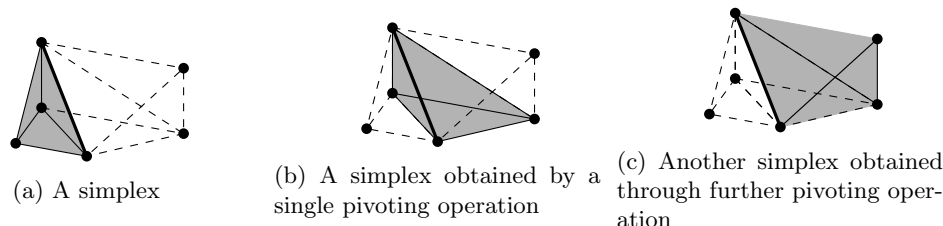


Fig. 5. Via the pivoting procedure, one moves from a simplex (a) to a different simplex (b) by leaving a chosen vertex and then to yet another simplex (c).

4.3. Traverse the feasible subgraph

In the above we have formulated the enumeration of lower d -faces as the problem of exploring the feasible subgraph of which the lower d -faces is a subset. We also have two procedures for “walking” within the graph: The extension procedure moves from one lower face to another of one higher dimension while the pivoting procedure jumps from one lower d -face to another lower d -face. With these building blocks in place, the exploration can be handled by classic graph traversal algorithms which we shall briefly review for completeness.

Most graph traversal algorithms follow a “discover-explore” procedure with proper book keeping [41]: They gradually explore the graph node by node through the connection

between them while keeping track of the nodes visited so that no node is explored twice. For a single node, such an algorithm is divided into the *discover* and *explore* stages: a node is first discovered, and then its connections to other yet unknown nodes are explored. Clearly, each node only needs to be visited once. That is, one only needs to explore a *spanning tree* of the graph, (a subgraph that contains all the vertices but is a tree in structure), so some mechanism must be used to prevent a node from being visited twice. To keep track of the nodes as they are being visited, we assign each task a dynamic marker – its state. A node can be in one of the following three states:

undiscovered The initial status of every node. In this state, the existence of the node is completely unknown to us.

discovered The existence of the node is known, but its connections to other nodes are not yet explored.

completely-explored The existence of the node is known and its connections to other nodes have been fully explored.

Obviously, a node cannot be *completely-explored* before it is first discovered, so in the course of the algorithm, the state of vertices progresses from *undiscovered* to *discovered* to *completely-explored*. This point of view also reveals the parallelism in such algorithms: nodes on different branches of the spanning tree can be explored in parallel, while consecutive nodes on a single branch must be discovered and explored in order. To start the algorithm, an initial set of nodes are generated by some other means (bootstrapping). The algorithm then discovers other nodes through their connections. From these newly discovered nodes the algorithm can discover even more node. This will continue as a self-sustaining process until all connected vertices are visited.

A complete algorithm also need a data structure to keep track of the discovered but not yet completely explored vertices (bookkeeping). In the present work, a *queue* is used. It is a linear data structure where newly discovered nodes are added to the back-end of the queue. The use of the queue structure essentially imposes an implicit ordering of “first-in-first-out”, that is, nodes discovered first are explored first. In the context of graph traversal algorithms, this is referred to as a *breadth-first* strategy in exploring the feasible subgraph. Experiments, presented in [5], suggest that a more flexible ordering of nodes within the queue may provide better performance, scalability, and memory usage. However, for simplicity, in this work, only the breadth-first approach has been studied. The detail of this class of algorithms can be found in standard textbooks such as [41]. In §4.5, we list the pseudo code.

4.4. Checking for duplicated discovery

One important problem we must deal with, in the parallel algorithm, is that same nodes may be discovered by different threads at the same time. Since the degree is the sum of the normalized volume of the projection of all the lower d -faces which are represented by the nodes in the feasible subgraph, duplicated nodes will produce incorrect results. Therefore, the mechanism for ensuring no duplicated lower d -faces are listed is the key to the correctness of the algorithm.

This mechanism appears to be the bottleneck, in terms of performance, of the original algorithm [14] for enumerating “mixed cells” using mainly the pivoting process which is one of the main inspiration of the present work (see Remark 2). Our experiments confirm that an inefficient checking mechanism would be the limiting factor of the scalability in

a parallel implementation. Since on a GPU, it is typical to have thousands of threads active simultaneously, the efficiency of such a mechanism is crucial.

In the present work, the *hash table* data structure is used to keep track of the nodes, in the graph, that have been discovered or completely-explored. The great advantage of this choice is that unlike a sorted data structure, hash table provides nearly constant access time, in *most* cases. In our current implementation, for simplicity, the well-tested bit-string hash function from the standard C++ library is used.

Our experiments suggest that a hash table with 2^{16} to 2^{20} entries is sufficient for all problems considered in §6 in the sense that the collision rate in hash table access can be virtually ignored.

4.5. Summary of the algorithm

In the above, we formulate the degree computation for solution components defined by binomial systems as the exploration of the feasible subgraph to be accomplished by the two complementing processes: extension and pivoting. In this section, we list the main algorithms.

These algorithms are designed for a system with one or more GPU devices and a single CPU with the GPU performance most of the computation intensive tasks. For simplicity, we restrict ourselves to modern GPUs manufactured by NVidia and build our program based on NVidia CUDA (a GPU programming framework). All the GPU devices must share memory since they must all have access to data structures **WaitingNodes**, **KnownNodes**, and **NewNodes**. In the current implementation, this is accomplished via a technique known as *pinned memory* [40] provided by the CUDA framework.

In the following algorithms, the list **WaitingNodes** contains nodes whose feasibility are to be determined by the extension procedure. **Cells** is the unordered collection of lower d -faces already discovered. **KnownNodes** is the hash table that record the discovery of nodes, and it is crucial mechanism by which we ensure the uniqueness of the discovered nodes. Finally, **NewNodes** is an unordered list that keeps track of nodes discovered through pivoting or extension. They need to be checked against **KnownNodes** for uniqueness.

RANDOM is a function that randomly choose an item from a collection using pseudo random number generator. The randomness is employed to achieve a more uniform performance from one run to another which simplifies the benchmarking process. SIMPLEX-PHASEONE and SIMPLEXPHASETWO are the phase-one and phase-two algorithm of the simplex method for the linear programming problems (21) and (23) respectively. Even though, at over 3000 lines, the C++ code for these two components are the longest and most complicated parts of the entire program, they have been a fixture of the long line of “mixed volume computation” software developed over the last two decades whence the present work inherits much of its techniques and design. Therefore we choose to not describe them in detail and refer to works including [5, 14, 15, 28, 30, 38, 39].

The EXTEND procedure tests the feasibility of a node (see §4.1) in the waiting list **WaitingNodes**, and it is designed to run simultaneously on all available threads across all GPU devices.

```

1: function EXTEND
2:   if WaitingQueue  $\neq \emptyset$  then
3:      $\{a_0, \dots, a_k\} \leftarrow \text{DEQUEUE}(\text{WaitingQueue})$ 
4:      $F \leftarrow \text{SIMPLEXPHASETWO}(LP(\{a_0, \dots, a_k\}))$ 

```



```

5:      if  $F \neq \emptyset$  then
6:           $\text{NewNodes} \leftarrow \text{NewNodes} \cup \{F\}$ 
7:      end if
8:  end if
9: end function

```

The PIVOT procedure implements the simplicial pivoting process detailed in §4.2, and it is designed to run simultaneously on all available threads across all GPU devices. It picks a random lower d -face already discovered and apply simplicial pivoting to potentially obtain a new lower faces. Just like the EXTEND procedure above, newly discovered nodes will be place in the NewNodes list.

```

1: function PIVOT
2:   if  $\text{Cells} \neq \emptyset$  then
3:        $\{a_0, \dots, a_d\} \leftarrow \text{RANDOM}(\text{Cells})$ 
4:        $\ell \leftarrow \min(d+1, 10)$ 
5:       for  $i = 1, \dots, \ell$  do
6:            $j \leftarrow \text{RANDOM}(\{0, \dots, d\})$ 
7:            $F \leftarrow \text{SIMPLEXPHASEONE}(P(\{a_0, \dots, a_d\} \setminus \{a_j\}))$ 
8:           if  $F \neq \emptyset$  then
9:                $\text{NewNodes} \leftarrow \text{NewNodes} \cup \{F\}$ 
10:          end if
11:       end for
12:   end if
13: end function

```

The procedure CHECKUNIQ checks newly discovered nodes against the hash table KnownNodes to make sure they have not already been discovered. It will run on a GPU device with a large number of threads simultaneously checking the uniqueness of all nodes in the list of NewNodes .

```

1: function CHECKUNIQ
2:   if  $\text{NewNodes} \neq \emptyset$  then
3:        $\{a_0, \dots, a_k\} \leftarrow \text{DEQUEUE}(\text{NewNodes})$ 
4:       if  $\{a_0, \dots, a_k\} \notin \text{KnownNodes}$  then
5:            $\text{KnownNodes} = \text{KnownNodes} \cup \{F\}$ 
6:           if  $k = d+1$  then
7:                $\text{Cells} = \text{Cells} \cup \{\{a_0, \dots, a_k\}\}$ 
8:           else
9:               for all  $a \in S \setminus \{a_0, \dots, a_k\}$  do
10:                  if  $\{a_0, \dots, a_k, a\} \notin \text{KnownNodes}$  then
11:                       $\text{WaitingQueue} = \text{WaitingQueue} \cup \{\{a_0, \dots, a_k, a\}\}$ 
12:                  end if
13:              end for
14:           end if
15:       end if
16:   end if
17: end function

```

Finally, the main procedure, which runs on the CPU, coordinates all the different processes.

```

1: function MAIN
2:   WaitingNodes  $\leftarrow S$ 
3:   while WaitingNodes  $\neq \emptyset$  do
4:     Run EXTEND on available GPU threads
5:     Run PIVOT on available GPU threads
6:     Wait for EXTEND and PIVOT
7:     Run CHECKUNIQ on available GPU threads
8:   end while
9: end function

```

5. Computation of witness sets

The concept of “witness sets” [42, 43] is one of the most fundamental and versatile tool in numerical algebraic geometry. In its most basic form, given a pure dimensional algebraic set, it can be shown that its intersection with a generic affine space of complementary dimension consists of finitely many isolated points. This finite set is called a *witness set* of the algebraic set. It can be used to compute, among many other things, the irreducible decomposition and primary decomposition numerically. In many scenarios, it produces the degree of each component as a byproduct. Indeed, this technique (via witness sets) was first used to numerically compute the degrees of the “Master Space” problem in the work [22].

Given the ubiquity of the use of witness sets in numerical algebraic geometry, in this section, we shall briefly outline a homotopy construction for computing witness sets for a component of $\mathcal{V}^*(\mathbf{x}^A - \mathbf{b})$. It is a special case of the *polyhedral homotopy* [24].

Recall that by Proposition 3, the intersection between a component $V \subseteq \mathcal{V}^*(\mathbf{x}^A - \mathbf{b})$ and a generic affine space of complementary dimension consists of precisely the points $\mathbf{t} = (t_1, \dots, t_d) \in (\mathbb{C}^*)^d$ that satisfy the system of d Laurent polynomial equation in d variables given by

$$\begin{aligned}
c_{11}\mathbf{t}^{\mathbf{p}_0^{(1)}} + c_{12}\mathbf{t}^{\mathbf{p}_0^{(2)}} + \dots + c_{1n}\mathbf{t}^{\mathbf{p}_0^{(n)}} &= c_{10} \\
&\vdots \\
c_{d1}\mathbf{t}^{\mathbf{p}_0^{(1)}} + c_{d2}\mathbf{t}^{\mathbf{p}_0^{(2)}} + \dots + c_{dn}\mathbf{t}^{\mathbf{p}_0^{(n)}} &= c_{d0}
\end{aligned} \tag{24}$$

where the coefficients depends on both the choice of the component in $\mathcal{V}^*(\mathbf{x}^A - \mathbf{b})$ and the choice of the r -dimensional affine space.

Reusing the notations from §4, let $S = \{\mathbf{p}_0^{(1)}, \dots, \mathbf{p}_0^{(n)}\}$, and let $\omega : S \rightarrow \mathbb{R}$ be the generic lifting function used for constructing regular simplicial subdivision of $\text{conv } S$ in §4. Without loss of generality, we can pick ω to have images only in \mathbb{Q} . With these, we introduce a new variable s and consider

$$H(\mathbf{t}, s) = \begin{cases} c_{11}\mathbf{t}^{\mathbf{p}_0^{(1)}} s^{\omega(\mathbf{p}_0^{(1)})} + \dots + c_{1n}\mathbf{t}^{\mathbf{p}_0^{(n)}} s^{\omega(\mathbf{p}_0^{(n)})} - c_{10}s^{\omega(\mathbf{p}_0^{(1)})} = \sum_{\mathbf{a} \in S} c_{1,\mathbf{a}} \mathbf{t}^{\mathbf{a}} s^{\omega(\mathbf{a})} \\ \vdots \\ c_{d1}\mathbf{t}^{\mathbf{p}_0^{(1)}} s^{\omega(\mathbf{p}_0^{(1)})} + \dots + c_{dn}\mathbf{t}^{\mathbf{p}_0^{(n)}} s^{\omega(\mathbf{p}_0^{(n)})} - c_{d0}s^{\omega(\mathbf{p}_0^{(1)})} = \sum_{\mathbf{a} \in S} c_{d,\mathbf{a}} \mathbf{t}^{\mathbf{a}} s^{\omega(\mathbf{a})} \end{cases} \tag{25}$$

which is constructed by multiplying each term in (24) by a rational power of the new variable s whose exponent is determined by the lifting function $\omega : S \rightarrow \mathbb{Q}$. Clearly, $H(\mathbf{t}, 1) = \mathbf{0}$ is exactly the system (24) which we aim to solve (inside $(\mathbb{C}^*)^d$). As s varies,

however, H represents a continuous deformation of the system (24), or a *homotopy*. The central idea behind the *homotopy continuation method* for solving systems of equations is the deformation of a system into a “starting system” which one can solve easily. Then numerical continuation methods are employed to trace the movement of the solutions of the starting system under the deformation toward the solutions of the original system which one aims to solve.

The key here is to find an appropriate starting system that can be easily solved. As is, $H(\mathbf{t}, 0)$ cannot be used as the starting system since at $s = 0$, the system is either identically zero or undefined. Therefore certain transformation is necessary to produce a meaningful and solvable starting system. Such transformations are given by the regular simplicial subdivision discussed in §4.

Still let \mathcal{D} be a regular simplicial subdivision obtained by the algorithm presented in §4.5. Recall that each cell in \mathcal{D} is a projection of a cell of the form $\{\hat{\mathbf{a}}_0, \dots, \hat{\mathbf{a}}_d\}$ such that $\text{conv}\{\hat{\mathbf{a}}_0, \dots, \hat{\mathbf{a}}_d\}$ is a lower d -face of $\text{conv} \hat{\mathcal{S}}$ that is characterized by (22). That is, there exists a (unique) vector of the form $\hat{\boldsymbol{\alpha}} = (\alpha_1, \dots, \alpha_d, 1)$ such that

$$\begin{aligned} \langle \hat{\mathbf{a}}_0, \hat{\boldsymbol{\alpha}} \rangle &= \langle \hat{\mathbf{a}}_j, \hat{\boldsymbol{\alpha}} \rangle & \text{for } j = 1, \dots, d \\ \langle \hat{\mathbf{a}}_0, \hat{\boldsymbol{\alpha}} \rangle &< \langle \hat{\mathbf{a}}, \hat{\boldsymbol{\alpha}} \rangle & \text{for all } \mathbf{a} \in S \end{aligned} \quad (26)$$

Using $\hat{\boldsymbol{\alpha}} = (\alpha_1, \dots, \alpha_d, 1)$, we shall consider the change of variables

$$\mathbf{t} = \begin{cases} t_1 &= y_1 s^{\alpha_1} \\ &\vdots \\ t_d &= y_d s^{\alpha_d} \end{cases} \quad (27)$$

with which H becomes

$$H(\mathbf{t}, s) = H(y_1 s^{\alpha_1}, \dots, y_d s^{\alpha_d}, s) = \begin{cases} \sum_{\mathbf{a} \in S} c_{1,\mathbf{a}} \mathbf{y}^{\mathbf{a}} s^{\langle \mathbf{a}, \boldsymbol{\alpha} \rangle + \omega(\mathbf{a})} = \sum_{\mathbf{a} \in S} c_{1,\mathbf{a}} \mathbf{y}^{\mathbf{a}} s^{\langle \hat{\mathbf{a}}, \hat{\boldsymbol{\alpha}} \rangle} \\ \vdots \\ \sum_{\mathbf{a} \in S} c_{d,\mathbf{a}} \mathbf{y}^{\mathbf{a}} s^{\langle \mathbf{a}, \boldsymbol{\alpha} \rangle + \omega(\mathbf{a})} = \sum_{\mathbf{a} \in S} c_{d,\mathbf{a}} \mathbf{y}^{\mathbf{a}} s^{\langle \hat{\mathbf{a}}, \hat{\boldsymbol{\alpha}} \rangle} \end{cases}$$

Let $\beta = \langle \hat{\mathbf{a}}_0, \hat{\boldsymbol{\alpha}} \rangle$ and define a new homotopy

$$H^{\boldsymbol{\alpha}, \beta}(\mathbf{y}, s) = s^{-\beta} H(y_1 s^{\alpha_1}, \dots, y_d s^{\alpha_d}, s) = \begin{cases} s^{-\beta} \sum_{\mathbf{a} \in S} c_{1,\mathbf{a}} \mathbf{y}^{\mathbf{a}} s^{\langle \hat{\mathbf{a}}, \hat{\boldsymbol{\alpha}} \rangle} \\ \vdots \\ s^{-\beta} \sum_{\mathbf{a} \in S} c_{d,\mathbf{a}} \mathbf{y}^{\mathbf{a}} s^{\langle \hat{\mathbf{a}}, \hat{\boldsymbol{\alpha}} \rangle} \end{cases}$$

Note that the new homotopy still has the necessary property that $H^{\boldsymbol{\alpha}, \beta}(\mathbf{y}, 1) = \mathbf{0}$ is identical to the system (24) which we aim to solve.

One important observation here is that, by (26), there are precisely $d+1$ terms in each component of $H^{\boldsymbol{\alpha}, \beta}(\mathbf{y}, s)$ having no power of s (the terms corresponding to $\mathbf{a}_0, \dots, \mathbf{a}_d$), and all other terms have positive powers of s . Consequently, at $s = 0$, terms with positive

powers of s vanish, leaving only

$$\begin{cases} c_{1,\mathbf{a}_0} \mathbf{y}^{\mathbf{a}_0} + c_{1,\mathbf{a}_1} \mathbf{y}^{\mathbf{a}_1} + \cdots + c_{1,\mathbf{a}_d} \mathbf{y}^{\mathbf{a}_d} = 0 \\ c_{2,\mathbf{a}_0} \mathbf{y}^{\mathbf{a}_0} + c_{2,\mathbf{a}_1} \mathbf{y}^{\mathbf{a}_1} + \cdots + c_{2,\mathbf{a}_d} \mathbf{y}^{\mathbf{a}_d} = 0 \\ \vdots \\ c_{d,\mathbf{a}_0} \mathbf{y}^{\mathbf{a}_0} + c_{d,\mathbf{a}_1} \mathbf{y}^{\mathbf{a}_1} + \cdots + c_{d,\mathbf{a}_d} \mathbf{y}^{\mathbf{a}_d} = 0 \end{cases} \quad (28)$$

To simplify the notation, let

$$C = \begin{pmatrix} c_{1,\mathbf{a}_0} & \cdots & c_{1,\mathbf{a}_d} \\ \vdots & \ddots & \vdots \\ c_{d,\mathbf{a}_0} & \cdots & c_{d,\mathbf{a}_d} \end{pmatrix} \quad \Gamma = (\mathbf{a}_0 \cdots \mathbf{a}_d)$$

then the above equation can be written as

$$C \cdot (\mathbf{y}^\Gamma)^\top = \mathbf{0}. \quad (29)$$

For generic choices of the coefficients, there exists a nonsingular matrix $G \in M_{d \times d}(\mathbb{C})$ such that

$$GC = \begin{pmatrix} c_{11}^* & & c_{12}^* \\ & c_{21}^* & c_{22}^* \\ & & \ddots & \vdots \\ & & & c_{d1}^* & c_{d2}^* \end{pmatrix}. \quad (30)$$

for some $c_{ij}^* \in \mathbb{C}^*$. Then without altering its solution set, (29) can be transformed into the equivalent system

$$GC(\mathbf{t}^\Gamma)^\top = \begin{cases} c_{11}^* \mathbf{y}^{\mathbf{a}_0} & + c_{12}^* \mathbf{y}^{\mathbf{a}_d} = 0 \\ c_{21}^* \mathbf{y}^{\mathbf{a}_1} & + c_{22}^* \mathbf{y}^{\mathbf{a}_d} = 0 \\ \vdots & \vdots \\ c_{d1}^* \mathbf{y}^{\mathbf{a}_{d-1}} & + c_{d2}^* \mathbf{y}^{\mathbf{a}_d} = 0 \end{cases} \quad (31)$$

which is also a Laurent binomial system. Therefore, the algorithm outlined can then be used to solve this system. The solutions are precisely the solutions of the starting system (28) for the homotopy $H^{\alpha,\beta}$. Then numerical continuation techniques can be applied to trace the solutions toward $s = 1$ producing solutions to the target system (24), which will be points in the witness set of the component V of $\mathcal{V}^*(\mathbf{x}^A - b)$.

Recall that the construction of the homotopy $H^{\alpha,\beta}$ depends on a cell in the regular simplicial subdivision \mathcal{D} of $\text{conv } S$. It is typical for \mathcal{D} to contain more than one cell. In this case, each cell induce a different homotopy of the form of $H^{\alpha,\beta}$. The above construction is a special case of the *polyhedral homotopy* [24], and its theory guarantees that as one go through all the cells in \mathcal{D} , the resulting homotopies of the form $H^{\alpha,\beta}$ will find all the points in the witness set of V .

6. Master space of $\mathcal{N} = 1$ gauge theories

In this section, we consider a system arising from theoretical physics, in particular, string theory. A central area of current research in string theory is the study of the vacuum moduli space, which, roughly speaking, is the space of continuous solutions (or the affine algebraic variety) of a multivariate nonlinear function, called the superpotential of the theory under consideration. Here, the vacuum moduli spaces are spaces of special holonomy such as Calabi-Yau or G_2 manifolds. Different positive-dimensional components of the vacuum moduli space correspond to different particle branches, such as mesonic, baryonic, etc. Symbolic algebraic geometry methods have been used to study the complicated structures of the vacuum moduli spaces of various string theory models [17, 18, 19]. However, the methods are known to run out of the steam for even moderate sized systems due to the algorithmic complexity issues. Recently, numerical algebraic geometry methods have been introduced to string theory research and have solved bigger systems [20, 22, 23, 32, 33, 34, 35, 36].

In this article, we consider special types of models coming from string theory in which the systems to be solved are binomial systems, and in which the vacuum moduli spaces are composed of unions of positive-dimensional components. We take a model which is actively investigated by string theorists because its vacuum moduli space is a combination of mesonic and baryonic branches [10, 11]. Such moduli spaces are called *master spaces*.

In particular, we consider the superpotential for $\mathcal{N} = 1$ gauge theories for a $D3$ -brane on the Abelian orbifold $\mathbb{C}^3/\mathbb{Z}_m \times \mathbb{Z}_k$. The superpotential for this theory, for fixed $m, k \in \mathbb{N}$, is given by

$$W_{m,k} = \sum_{i=0}^{m-1} \sum_{j=0}^{k-1} x_{i,j} y_{i+1,j} z_{i+1,j+1} - y_{i,j} x_{i,j+1} z_{i+1,j+1} \quad (32)$$

where the periodic boundary conditions are imposed, e.g., $x_{i,m} = x_{i,0}$ for any i and $x_{k,j} = x_{0,j}$ for any j . This is a polynomial in $3mk$ variables: $x_{i,j}, y_{i,j}, z_{i,j}$ for the combinations of $i \in \mathbb{Z}_k$ and $j \in \mathbb{Z}_m$. For example, when $m = k = 2$, the superpotential is

$$\begin{aligned} W_{2,2} = & x_{0,0} y_{1,0} z_{1,1} - y_{0,0} x_{0,1} z_{1,1} + x_{0,1} y_{1,1} z_{1,0} - y_{0,1} x_{0,0} z_{1,0} \\ & + x_{1,0} y_{0,0} z_{0,1} - y_{1,0} x_{1,1} z_{0,1} + x_{1,1} y_{0,1} z_{0,0} - y_{1,1} x_{1,0} z_{0,0}, \end{aligned}$$

a polynomial in 12 variables $x_{0,0}, x_{0,1}, x_{1,0}, x_{1,1}, y_{0,0}, y_{0,1}, y_{1,0}, y_{1,1}, z_{0,0}, z_{0,1}, z_{1,0}, z_{1,1}$. We are interested in finding the critical points of $W_{m,k}$, that is, points at which all the partial derivatives of the superpotential $W_{m,k}$, with respect to variables $x_{i,j}, y_{i,j}, z_{i,j}$, are zero. These points are precisely the solutions to the system of polynomial equation

$$\frac{\partial W_{m,k}}{\partial x_{i,j}} = \frac{\partial W_{m,k}}{\partial y_{i,j}} = \frac{\partial W_{m,k}}{\partial z_{i,j}} = 0 \quad (33)$$

in the variables $x_{i,j}, y_{i,j}, z_{i,j}$.

Notice that in $W_{m,k}$, each variable appears in exactly two distinct terms. Consequently, the partial derivative of $W_{m,k}$ with respect to each variable consists of exactly two terms, hence it forms a binomial polynomial. For instance,

$$\frac{\partial W_{2,2}}{\partial x_{0,0}} = y_{1,0} z_{1,1} - y_{0,1} z_{1,0}, \quad \frac{\partial W_{2,2}}{\partial x_{0,1}} = -y_{0,0} z_{1,1} + y_{1,1} z_{1,0}.$$

m/k	1	2	3	4	5	6	7	8
1	N/A	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18
3	5	8	11	14	17	20	23	26
4	6	10	14	18	22	26	30	34
5	7	12	17	22	27	32	37	42
6	8	14	20	26	32	38	44	50
7	9	16	23	30	37	44	51	58
8	10	18	26	34	42	50	58	66

Table 2. The dimension of $\mathcal{V}^*(\nabla W_{m,k})$ for a range of values for m and k .

$m = k$	9	10	11	12	13	14	15	20	25	30	35	40
Dim.	83	102	123	146	171	198	227	402	627	902	1227	1602

Table 3. The dimension of $\mathcal{V}^*(\nabla W_{m,k})$ for a range of larger values for $m = k$.

Therefore (33) is indeed a binomial system which shall simply be denoted by $\nabla W_{m,k}$. We are interested in computing the dimension and degree of components of the \mathbb{C}^* -solution set $\mathcal{V}^*(\nabla W_{m,k})$ of this system.

The dimension and the degree of the top dimensional components of this system was first computed in [11] for up to $m = 3$ and $k = 5$ using the Gröbner basis method. Later on, in [36], the dimensions and degrees of all the components for up to $m = 3 = k$ were carried out using numerical algebraic geometry methods. The parallel GPU-based implementation of the binomial solver we have proposed can compute, very quickly, the dimension and the global parametrization of the \mathbb{C}^* -solution set for higher values of m and k . Table 2 and 3 show the dimension of $\mathcal{V}^*(\nabla W_{m,k}) \subset (\mathbb{C}^*)^{3mk}$ for a range of values for m and k . More importantly, our implementation shows impressive efficiency in computing the degree of $\mathcal{V}^*(\nabla W_{m,k})$ for larger values of m and k , including a component of degree as high as 50467100, for $m = 4$ and $k = 5$. Table 4 shows the degree of $\mathcal{V}^*(\nabla W_{m,k})$ for a range of m and k values which is a significant expansion of the existing results presented in [11, 36] and a substantial improvement over the existing algorithm outlined in [6].

Table 5 shows the speedup ratio achieved by the GPU-based algorithm, presented in §4, over its closest CPU-based implementation MixedVol-2.0 [28] which is widely regarded as one of fastest serial software program for computing “mixed volume” (See Remark 2 and Appendix A for its connection with degree computation considered in this article). Remarkably, with sufficient GPU threads nearly 30 fold speedup ratio has been achieved by the double-precision version of the algorithm. When the single-precision version is used, even higher speedup ratio can be achieved, Unfortunately, it appears that single-precision is, in general, not reliable in handling very large problems due to its insufficient precision.

More important to note is the great potential of the GPU-based algorithm when multiple GPU devices are used. Table 6 shows the speedup ratio achieved by multiple GPU devices when compared to a single GPU, a single CPU, and a small cluster of 100 nodes.

m/k	1	2	3	4	5	6	7	8
1		2	4	8	16	32	64	128
2	2	14	92	584	3632	22304	135872	823424
3	4	92	1620	26762	437038	7029180	111135118*	≥ 100100328
4	8	584	26762	1169876	50467100	≥ 11907022	≥ 37567994	
5	16	3632	437038	50467100	≥ 99710106	≥ 62944504		
6	32	22304	7029180	≥ 11907022	≥ 62944504			
7	64	135872	111135118*	≥ 37567994				
8	128	823424	≥ 100100328					

Table 4. The degree of the \mathbb{C}^* -solution set defined by (32) for a range of m and k values. This table lists only the results that can be computed within *1 hour* on a NVidia GTX 780 graphics card with the double-precision version of the GPU-based parallel algorithm presented in this work. Shaded entries correspond to and agree with the results already presented in [11, 36]. Entries marked by * are results that cannot be computed with any CPU-based program within a reasonable amount of time (2 days for multi-core systems and 7 days for clusters). Entries marked by \geq are lower bounds of the degrees computed by counting the total number of cells in the simplicial subdivision of the polytope associated with (32).

GPU threads	DP Speedup ratio	SP Speedup ratio
64	0.00	0.00
128	0.00	0.00
256	0.00	0.00
512	0.91	0.73
1024	0.98	4.14
2048	1.15	6.66
4096	2.20	10.99
8192	4.01	18.71
2^{14}	7.99	35.00
2^{15}	15.00	40.10
2^{16}	16.33	45.33
2^{17}	29.47	44.99
2^{18}	28.33	41.06

Table 5. Speedup ratios achieved by the GPU-based double-precision (DP) and single-precision (SP) algorithm respectively on NVidia GTX 780 when compared to MixedVol-2.0. “0.00” represents speedup ratios too small to be measured reliably. The number of threads are chosen to be multiples of 32 which is the “warp size” (smallest group of threads in CUDA framework).

With three GPU devices, over 60 fold speedup over the single-threaded CPU-based algorithm (MixedVol-2.0) has been achieved. The most surprising result is the comparison between the GPU-based algorithm, developed in this article, running on three GPU devices and a similar CPU-based algorithm running on a small cluster. MixedVol-3 is

N.o. devices	1	2	3
Speedup over single device	100%	188%	213%
Max. speedup over CPU	28.33	54.12	61.00
Max. speedup over a cluster of 100 nodes	0.48	0.91	1.04

Table 6. Speedup ratios achieved by using multiple identical NVidia GTX 780 devices with the single device performance (using the same algorithm) as a reference.

a parallel version of MixedVol-2.0 [28] and, now, a part of a larger software program Hom4PS-3 [4]. With three NVidia GTX 780 our GPU-based algorithm computes the degree for $\mathcal{V}^*(\nabla W_{4,5})$ faster than MixedVol-3 on a small cluster totaling 100 Intel Xeon 2.4Ghz processor cores.

In computing the degrees of $\mathcal{V}^*(\nabla W_{m,k})$ for certain larger m and k , while the GPU based algorithm was able to obtain the simplicial subdivisions of the polytopes associated with $\mathcal{V}^*(\nabla W_{m,k})$, the volumes of certain cells, which are given as a matrix determinant (16), could not be computed with sufficient accuracy to ensure the exactness of the answer. However, since the volume of each cell is at least one, the total number of cells is therefore a lower bound of the degree which equals the total volume of all the cells. Although these lower bounds are likely to be much smaller than the actual degrees, given the sheer size of these systems, these partial results still merit further investigations and improvements on the approach presented here. The lower bounds are therefore also included in Table 4 (entries marked with “ \geq ”).

While the rigorous analysis and physical interpretation of the data presented here are outside the scope of this article, the rich set of data shown in Tables 2, 3, and 4 appear to show some general pattern. To motivate further research in this important problem, we summarize these patterns in the form of a conjecture:

Conjecture 1. In general, for $m, k \in \mathbb{Z}^+$ with $m \neq k$, the solution set $\mathcal{V}^*(\nabla W_{m,k})$ consists of a single component of dimension

$$\dim \mathcal{V}^*(\nabla W_{m,k}) = mk + 2.$$

Furthermore, for $m = 1$ and $m = 2$, the degree of the solution set is given by

$$\begin{aligned} \deg \mathcal{V}^*(\nabla W_{1,k}) &= 2 \cdot \deg \mathcal{V}^*(\nabla W_{1,k-1}) = 2^{k-1} \\ \deg \mathcal{V}^*(\nabla W_{2,k}) &= 6 \cdot \deg \mathcal{V}^*(\nabla W_{2,k-1}) + 2^{2k-3} = 2 \cdot 6^{k-1} + \sum_{j=0}^{k-2} 2^{2(k-j)-3} \cdot 6^j \end{aligned}$$

7. Conclusion

This this article, we proposed a parallel algorithm for computing the degree of components of a \mathbb{C}^* -solution set defined by a binomial system that is specifically designed for GPU devices. Numerical experiments with the CUDA based implementation shows remarkable performance and scalability when applied to the binomial systems of the master space of $\mathcal{N} = 1$ gauge theories.

Acknowledgement

DM was supported by a DARPA Young Faculty Award and an Australian Research Council DECRA fellowship. TC was supported in part by NSF under Grant DMS 11-15587. TC and DM would like to thank Daniel Brake, Yang-Hui He and Thomas Kahle for their feedback on this paper. TC would also like to thank Dirk Colbry for the helpful discussions and the Institute for Cyber-Enabled Research at Michigan State University for providing the necessary hardware and computational infrastructure.

Appendix A Kushnirenko’s theorem

Theorem 1 (Kushnirenko [27]). Consider the system of k Laurent polynomial equations

$$\begin{cases} c_{1,1}\mathbf{x}^{\mathbf{a}^{(1)}} + c_{1,2}\mathbf{x}^{\mathbf{a}^{(2)}} + \cdots + c_{1,k}\mathbf{x}^{\mathbf{a}^{(\ell)}} &= 0 \\ c_{2,1}\mathbf{x}^{\mathbf{a}^{(1)}} + c_{2,2}\mathbf{x}^{\mathbf{a}^{(2)}} + \cdots + c_{2,k}\mathbf{x}^{\mathbf{a}^{(\ell)}} &= 0 \\ \vdots & \\ c_{k,1}\mathbf{x}^{\mathbf{a}^{(1)}} + c_{k,2}\mathbf{x}^{\mathbf{a}^{(2)}} + \cdots + c_{k,k}\mathbf{x}^{\mathbf{a}^{(\ell)}} &= 0 \end{cases}$$

in k variables $\mathbf{x} = (x_1, \dots, x_k)$ in which every equation has the same set of monomials determined by exponent vectors $\mathbf{a}^{(1)}, \dots, \mathbf{a}^{(\ell)} \in \mathbb{Z}^k$. With “generic” coefficients $c_{i,j} \in \mathbb{C}^*$, the solutions of this system in $(\mathbb{C}^*)^k$ are all isolated and nonsingular. The total number of these isolated solutions is

$$k! \cdot \text{Vol}_k(\text{conv}\{\mathbf{a}^{(1)}, \dots, \mathbf{a}^{(\ell)}\}).$$

This important result was later generalized significantly in [1] where the number of isolated nonzero \mathbb{C}^* -solutions of a system of Laurent polynomial equation is shown to be equal to the *mixed volume* of the *Newton polytopes* of the system. This number is now commonly known as the *BKK bound* of a Laurent polynomial system. Therefore, the degree computation discussed above can be considered as a special case of the BKK bound i.e. mixed volume computation.

References

- [1] D. N. Bernshtein. The number of roots of a system of equations. *Functional Analysis and its Applications*, 9(3):183–185, 1975.
- [2] I. Brny and Z. Fredi. Computing the volume is difficult. *Discrete & Computational Geometry*, 2(1):319–326, Dec. 1987.
- [3] B. Beler, A. Enge, and K. Fukuda. Exact volume computation for polytopes: A practical study. In G. Kalai and G. M. Ziegler, editors, *Polytopes Combinatorics and Computation*, number 29 in DMV Seminar, pages 131–154. Birkhuser Basel, Jan. 2000.
- [4] T. Chen, T.-L. Lee, and T.-Y. Li. Hom4PS-3: A parallel numerical solver for systems of polynomial equations based on polyhedral homotopy continuation methods. In H. Hong and C. Yap, editors, *Mathematical Software ICMS 2014*, number 8592 in Lecture Notes in Computer Science, pages 183–190. Springer Berlin Heidelberg, Jan. 2014.

- [5] T. Chen, T.-L. Lee, and T.-Y. Li. Mixed volume computation in parallel. *Taiwanese Journal of Mathematics*, 18(1):93–114, 2014.
- [6] T. Chen and T.-Y. Li. Solutions to systems of binomial equations. *Annales Mathematicae Silesianae*, 28:7–34, 2014.
- [7] H. Cohen. *A Course in Computational Algebraic Number Theory*. Springer Science & Business Media, Jan. 1993.
- [8] D. A. Cox, J. B. Little, and H. K. Schenck. *Toric varieties*. American Mathematical Soc., 2011.
- [9] D. Eisenbud and B. Sturmfels. Binomial ideals. *Duke Mathematical Journal*, 84(1):1–46, July 1996.
- [10] D. Forcella, A. Hanany, Y.-H. He, and A. Zaffaroni. Mastering the Master Space. *Lett.Math.Phys.*, 85:163–171, 2008.
- [11] D. Forcella, A. Hanany, Y.-H. He, and A. Zaffaroni. The Master Space of N=1 Gauge Theories. *JHEP*, 0808:012, 2008.
- [12] W. Fulton. *Introduction to toric varieties*. Number 131. Princeton University Press, 1993.
- [13] W. Fulton. *Intersection Theory*. Springer New York, Jan. 1998.
- [14] T. Gao and T. Y. Li. Mixed volume computation via linear programming. *Taiwanese Journal of Mathematics*, 4(4):pp. 599–619, Jan. 2000.
- [15] T. Gao and T.-Y. Li. Mixed volume computation for semi-mixed systems. *Discrete & Computational Geometry*, 29(2):257–277, Jan. 2003.
- [16] M. S. Gockenbach. *Finite-Dimensional Linear Algebra*. CRC Press, June 2011.
- [17] J. Gray. A Simple Introduction to Grobner Basis Methods in String Phenomenology. *Adv.High Energy Phys.*, 2011, 19??
- [18] J. Gray, Y.-H. He, A. Ilderton, and A. Lukas. STRINGVACUA: A Mathematica Package for Studying Vacuum Configurations in String Phenomenology. *Comput. Phys. Commun.*, 180:107–119, 2009.
- [19] J. Gray, Y.-H. He, and A. Lukas. Algorithmic Algebraic Geometry and Flux Vacua. *JHEP*, 0609:031, 2006.
- [20] B. Greene, D. Kagan, A. Masoumi, D. Mehta, E. J. Weinberg, and X. Xiao. Tumbling through a landscape: Evidence of instabilities in high-dimensional moduli spaces. *Phys.Rev.*, D88(2):026005, 2013.
- [21] R. Hartshorne. *Algebraic geometry*. Number 52. Springer, 1977.
- [22] J. Hauenstein, Y.-H. He, and D. Mehta. Numerical elimination and moduli space of vacua. *JHEP*, 1309:083, 2013.
- [23] Y.-H. He, D. Mehta, M. Niemerg, M. Rummel, and A. Valeanu. Exploring the Potential Energy Landscape Over a Large Parameter-Space. *JHEP*, 1307:050, 2013.
- [24] B. Huber and B. Sturmfels. A polyhedral method for solving sparse polynomial systems. *Mathematics of computation*, 64(212):1541–1555, 1995.
- [25] T. Kahle. Decompositions of binomial ideals. *Annals of the Institute of Statistical Mathematics*, 62(4):727–745, 2010.
- [26] T. Kahle and E. Miller. Decompositions of commutative monoid congruences and binomial ideals. *Algebra & Number Theory*, 8(6):1297–1364, 2014.
- [27] A. G. Kushnirenko. A newton polyhedron and the number of solutions of a system of k equations in k unknowns. *Usp. Math. Nauk*, 30:266–267, 1975.
- [28] T.-L. Lee and T.-Y. Li. Mixed volume computation in solving polynomial systems. *Contemp. Math*, 556:97–112, 2011.

- [29] T.-Y. Li. Numerical solution of polynomial systems by homotopy continuation methods. In P. G. Ciarlet, editor, *Handbook of Numerical Analysis*, volume 11, pages 209–304. North-Holland, 2003.
- [30] T. Y. Li and X. Li. Finding mixed cells in the mixed volume computation. *Foundations of Computational Mathematics*, 1(2):161–181, Jan. 2001.
- [31] J. D. Loera, J. Rambau, and F. Santos. *Triangulations: Structures for Algorithms and Applications*. Springer Science & Business Media, Aug. 2010.
- [32] D. Martinez-Pedreria, D. Mehta, M. Rummel, and A. Westphal. Finding all flux vacua in an explicit example. *JHEP*, 1306:110, 2013.
- [33] D. Mehta. Lattice vs. Continuum: Landau Gauge Fixing and 't Hooft-Polyakov Monopoles. *Ph.D. Thesis, The Uni. of Adelaide, Australasian Digital Theses Program*, 2009.
- [34] D. Mehta. Finding All the Stationary Points of a Potential Energy Landscape via Numerical Polynomial Homotopy Continuation Method. *Phys.Rev.*, E84:025702, 2011.
- [35] D. Mehta. Numerical Polynomial Homotopy Continuation Method and String Vacua. *Adv.High Energy Phys.*, 2011:263937, 2011.
- [36] D. Mehta, Y.-H. He, and J. D. Hauenstein. Numerical Algebraic Geometry: A New Perspective on String and Gauge Theories. *JHEP*, 1207:018, 2012.
- [37] E. Miller and B. Sturmfels. *Combinatorial commutative algebra*, volume 227. Springer, 2005.
- [38] T. Mizutani and A. Takeda. DEMiCs: A software package for computing the mixed volume via dynamic enumeration of all mixed cells. In M. Stillman, J. Verschelde, and N. Takayama, editors, *Software for Algebraic Geometry*, number 148 in The IMA Volumes in Mathematics and its Applications, pages 59–79. Springer, Jan. 2008.
- [39] T. Mizutani, A. Takeda, and M. Kojima. Dynamic enumeration of all mixed cells. *Discrete & Computational Geometry*, 37(3):351–367, Mar. 2007.
- [40] NVIDIA Corporation. NVIDIA CUDA c programming guide. Technical report, July 2011.
- [41] S. S. Skiena. *The Algorithm Design Manual*. Springer Science & Business Media, Apr. 2009.
- [42] A. J. Sommese, J. Verschelde, and C. W. Wampler. Numerical decomposition of the solution sets of polynomial systems into irreducible components. *SIAM Journal on Numerical Analysis*, 38(6):2022–2046, 2001.
- [43] A. J. Sommese and C. W. Wampler. *The Numerical solution of systems of polynomials arising in engineering and science*. World Scientific Pub Co Inc, 2005.
- [44] B. Sturmfels. Equations defining toric varieties. In *PROC. SYMPOSIA IN PURE*. Citeseer, 1997.
- [45] J. Verschelde, K. Gatermann, and R. Cools. Mixed-volume computation by dynamic lifting applied to polynomial system solving. *Discrete & Computational Geometry*, 16(1):69–112, Jan. 1996.